APPM 4650 | HW 4 | Soroush Khadem

1.) a.) $f(x) = \frac{1}{1-x}$

i.) $K_f = \left|\frac{x}{f(x)} \cdot f'(x)\right| = \left|x \cdot (1-x) \cdot \frac{1}{(1-x)^2}\right| = \left|\frac{x}{(1-x)}\right|$

if $x \approx 1 \Rightarrow x = 1+\varepsilon$: $K_f = \left|\frac{1+\varepsilon}{\varepsilon}\right|$ which can be large if ε is small enough $\Rightarrow$ ill conditioned

ii.) $x = 1-10^{-13}$: $K_f = \left|\frac{1-10^{-13}}{10^{-13}}\right| \Rightarrow$ lose approx. 13 digits, so expect 3 correct digits (see notebook.)

iii.) See notebook

iv.) $K_f = \left|\frac{1+1.13\times10^{-13}}{1.13\times10^{-13}}\right| \Rightarrow$ expect to lose 13 digits again. But, lose more (see notebook)

b.) $f(x) = \frac{\frac{c}{1-x}+d}{\frac{1}{1-x}+1}$, $c \neq 0$.

$$K_f = \left|\frac{x}{f(x)} \cdot f'(x)\right| = \left|\frac{x \cdot \left(\frac{1}{1-x}+1\right)}{\frac{c}{1-x}+d} \cdot \frac{d}{dx} \frac{\frac{c}{1-x}+d}{\frac{1}{1-x}+1}\right|$$

$$= \left|\frac{\frac{x}{1-x}+x}{\frac{c}{1-x}+d} \cdot \frac{\left(\frac{1}{1-x}+1\right)\left(\frac{c}{(1-x)^2}\right)-\left(\frac{c}{1-x}+d\right)\left(\frac{1}{(1-x)^2}\right)}{\left(\frac{1}{1-x}+1\right)^2}\right|$$

$$= \left|\frac{\frac{2x-x^2}{\cancel{1-x}}}{\frac{c+d-dx}{\cancel{1-x}}} \cdot \frac{\frac{c}{(1-x)^3}+\frac{c}{(1-x)^2}-\frac{c}{(1-x)^3}-\frac{d}{(1-x)^2}}{\left(\frac{2-x}{1-x}\right)^2}\right|$$

$$= \left|\frac{2x-x^2}{c+d-dx} \cdot \frac{c-d}{2-x}\right| , \quad K_f(x=1) = \frac{2(c-d)}{c} = 2-\frac{2d}{c}$$

hw4

October 4, 2020

```
In [1]: import numpy as np
```

1 Homework 4

1.1 Soroush Khadem

1.1.1 Problem 1

```
In [2]: f = lambda x : 1 / (1 - x)
```

```
In [3]: relAccuracy = lambda x, true : np.abs(x - true)/np.abs(true)
        numDigits   = lambda x, true : -np.log10( relAccuracy(x, true) + 1e-100 )
```

1.1.2 Part II

```
In [4]: numDigits(f(1 - 1e-13), 1e13)
```

```
Out[4]: 3.5074511814908544
```

This makes sense because it means that we lost 13 digits, which is what was predicted by the condition number

1.1.3 Part IV

```
In [5]: numDigits(f(1 - 1.13e-13), 1.13e13)
```

```
Out[5]: 0.6635467129687505
```

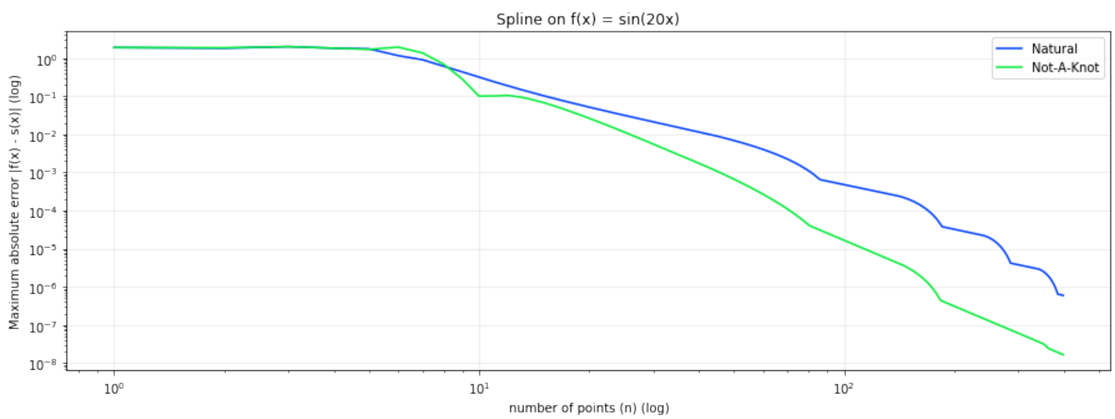
1.1.4 Problem 2

```
In [6]: from scipy import interpolate
        import matplotlib.pyplot as plt
        plt.style.use('seaborn-bright')
        plt.rcParams['figure.figsize'] = [15, 5]
        plt.rcParams['axes.grid'] = True
        plt.rcParams['grid.alpha'] = 0.25
```

```
In [7]: # The true function: sin(20x)
        f = lambda x : np.sin(20*x)
```

In [8]:
```
xs = np.random.uniform(low=1.01, high=1.99, size=(int(1e5),))
ns = np.arange(1, 400)
natural_error = []
not_knot_error = []
for n in ns:
    nodes = np.linspace(1, 2, num=n+1)
    natural_s = interpolate.CubicSpline(nodes, f(nodes), bc_type='natural')
    not_knot_s = interpolate.CubicSpline(nodes, f(nodes), bc_type='not-a-knot')
    natural_error.append(np.max(np.abs(f(xs) - natural_s(xs))))
    not_knot_error.append(np.max(np.abs(f(xs) - not_knot_s(xs))))
```

In [9]:
```
plt.plot(ns, natural_error)
plt.plot(ns, not_knot_error)
plt.xscale('log')
plt.yscale('log')
plt.title('Spline on f(x) = sin(20x)')
plt.xlabel('number of points (n) (log)')
plt.ylabel('Maximum absolute error |f(x) - s(x)| (log)')
plt.legend(['Natural','Not-A-Knot'])
plt.show()
```



1.1.5 Part B

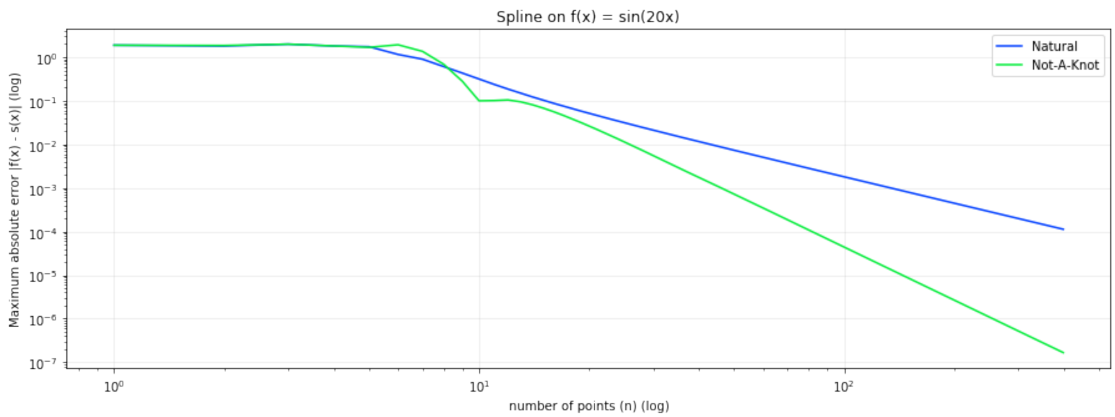
This is the convergence I expect, since the error approaches a straight line on a log log plot, and the convergence for a cubic cpline should be 4th order accurate.

1.1.6 Part C

In [10]:
```
xs = np.random.uniform(low=1, high=2, size=(int(1e5),))
ns = np.arange(1, 400)
natural_error = []
not_knot_error = []
for n in ns:
```

```
        nodes = np.linspace(1, 2, num=n+1)
        natural_s = interpolate.CubicSpline(nodes, f(nodes), bc_type='natural')
        not_knot_s = interpolate.CubicSpline(nodes, f(nodes), bc_type='not-a-knot')
        natural_error.append(np.max(np.abs(f(xs) - natural_s(xs))))
        not_knot_error.append(np.max(np.abs(f(xs) - not_knot_s(xs))))
```

```
In [11]: plt.plot(ns, natural_error)
         plt.plot(ns, not_knot_error)
         plt.xscale('log')
         plt.yscale('log')
         plt.title('Spline on f(x) = sin(20x)')
         plt.xlabel('number of points (n) (log)')
         plt.ylabel('Maximum absolute error |f(x) - s(x)| (log)')
         plt.legend(['Natural','Not-A-Knot'])
         plt.show()
```



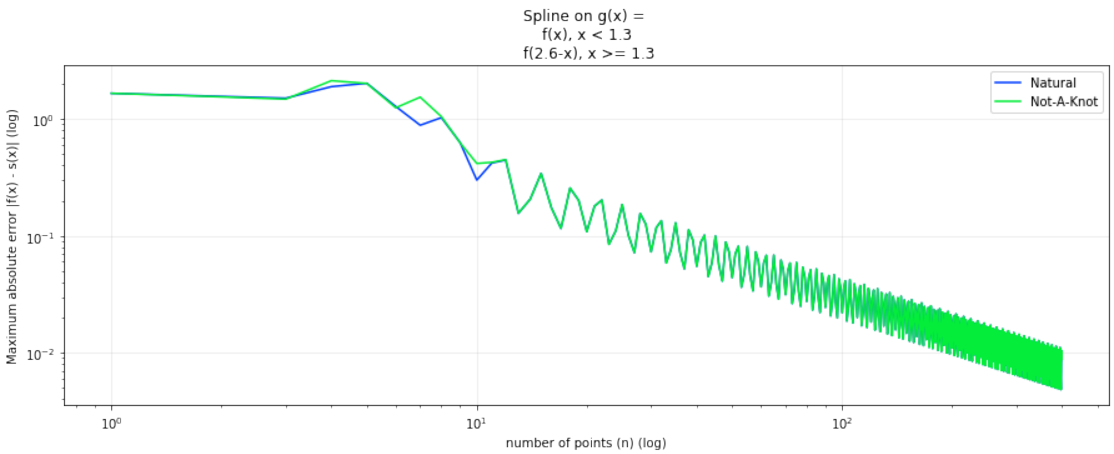
Because now the test points include the boundaries, the 'Not-A-Knot' method has a faster convergence, since it handles data on the end points more accurately, by specifying that the 3rd derivative should be 0 at the end points.

1.1.7 Part D

```
In [12]: g_func = lambda x : f(x) if x < 1.3 else f(2.6-x)
         g = np.vectorize(g_func)
```

```
In [13]: xs = np.random.uniform(low=1.01, high=1.99, size=(int(1e5),))
         ns = np.arange(1, 400)
         natural_error = []
         not_knot_error = []
         for n in ns:
             nodes = np.linspace(1, 2, num=n+1)
             natural_s = interpolate.CubicSpline(nodes, g(nodes), bc_type='natural')
             not_knot_s = interpolate.CubicSpline(nodes, g(nodes), bc_type='not-a-knot')
             natural_error.append(np.max(np.abs(g(xs) - natural_s(xs))))
             not_knot_error.append(np.max(np.abs(g(xs) - not_knot_s(xs))))
```

```
In [14]: plt.plot(ns, natural_error)
         plt.plot(ns, not_knot_error)
         plt.xscale('log')
         plt.yscale('log')
         plt.title('Spline on g(x) = \nf(x), x < 1.3\n f(2.6-x), x >= 1.3')
         plt.xlabel('number of points (n) (log)')
         plt.ylabel('Maximum absolute error |f(x) - s(x)| (log)')
         plt.legend(['Natural','Not-A-Knot'])
         plt.show()
```



Since the function is not differentiable on its whole domain (discontinuity at x = 1.3), the error of the cubic spline does not converge

```
In [ ]:
```